How Engineering Mathematics can Improve Software

David Lorge Parnas,

Middle Road Software, Ottawa, Ontario, Canada

Abstract

Abstract - For many decades computer science researchers have promised that the "Formal Methods" developed by computer scientists would bring about a drastic improvement in the quality and cost of software. That improvement has not materialized. We review the reasons for this failure. We then explain the difference between the notations that are used in formal methods and the mathematics that is essential in other areas of Engineering. Finally, we illustrate the ways that Engineering Mathematics can be useful in software projects

1. The Main Messages In This Talk ----3
2. Traditional Engineering compared to “Software Engineering” ----4
3. Early Ways to Apply Mathematics In Software Design ----5
4. Claims of Success/Progress for Formal Methods ----6
5. Role of mathematics in Engineering ----7
6. Roles of Documentation In Engineering ----8
7. Specifications vs. Other Descriptions ----9
8. Models vs. Documents ----10
9. Why I Stress Documentation ----11
10. Role of Mathematics in Today’s Software Development --12
11. Important Characteristics of Engineering Mathematics --13
12. What are Formal Methods ----14
13. Comparison: Engineering Math vs. Formal Methods ----15
14. Abstractions and Their Representation ----16
15. Mathematical Calculations ----17
16. Function Evaluation ----18
17. Predicate Satisfaction ----19
18. Inference or Deduction ----20
19. Mathematical Reasoning in Engineering. ----21
20. Applying Mathematics to Physical Systems ----22
21. Applying Mathematics ----23
22. How Engineering Mathematics Deals with Time ----24
23. Transfer Functions ----25
24. Applying Mathematics to Software Systems (I) ----26
25. Assertion Based Methods (Best Known) ----27
26. Predicate Transformer Methods ----28
27. Abstract Relational (Functional) Methods ----29
28. Additional views of software ----30
29. Applying Mathematics For Other Views ----31
30. Relational Content of Main Software Design Documents-32
31. Representing Software Relations ----33
32. Example of Conventional Expression ----34
33. Tabular Expression for Keyboard Checker ----35
34. Evaluating a Tabular Expression ----36
35. Why Tabular Expressions Make Mathematics Applicable-37
36. Defining The Meaning of Tabular Expressions ----38
37. Types of Tabular Expressions ----39
38. Promoting Discipline ----40
39. Knowing When to Stop ----41
40. So, How Should Software Engineers Use Mathematics? -42
41. Formal Methods, Who Needs Them? ----43
42. What Can We Do With Mathematics ----44
43. Back To The Main Messages ----45

The Main Messages In This Talk

1. CS Formal Methods have not succeeded and will not succeed: Why?
2. Engineering Mathematics is different from CS Formal Methods
3. Engineering Mathematics can be adapted for discrete functions
4. Mathematics can help those who build and maintain software

Traditional Engineering compared to “Software Engineering”

Two main differences:

- Much less physics
- Mathematics not used

One of these is understandable

- Components and products are not physical

One of these is not

- Mathematics can be helpful in many ways

Early Ways to Apply Mathematics In Software Design

Robert Floyd's pioneering work: Reducing issue of correctness to a mathematical theorems. Remaining work is mathematics

- Floyd, R.W., "*Assigning Meanings to Programs*", Proceedings of the Symposium of Applied Mathematics, Vol.19, 1968. Also in: Schwartz, J.T. (editor), Mathematical Aspects of Computer Science, American Mathematical Society, pp. 19-32, 1967.
- Hoare mixed proof and program analysis; introduced "Hoare triples"
- Dijkstra, E.W., "*A Discipline of Programming*", Prentice Hall, Englewood Cliffs, NJ, 1976. Introduced backward transformation of predicates.
- Mills, Harlan D.: "*The New Math of Computer Programming*", *Comm. ACM 18,*1, pp. 43-48 (1975) Program described by mapping from starting state to starting state.

Mills was the least inventive and the deepest mathematician.

Mills applied existing mathematics in a classical way.

- Function composition replaced predicate transformation.
- Search for proof was replaced by simplification.

Claims of Success/Progress for Formal Methods

Formal methods a very popular research area.

Funding agencies "encourages" industrial involvement and trials.

Trials succeed; why?

- Many smart people scrutinizing code in heroic efforts
- Assertions turn out to be debugging statements
- Same reader and writer (specifier and coder)
- Elovitz "pair programming" (ICSE 1979)

Most methods do not live past the experiment.

Publication is proof that the success is unusual.

Consider these reports with a "grain of salt".

If "a better mousetrap" was available, industry would clamour to use it.

Role of Mathematics In Engineering

- Documentation
  - Description
  - Specification
- Design
  - Parameter determination
  - Selection among alternatives
- Checks
  - Limit checking
  - Singularities
  - Completeness
  - Qualities (distortion, noise, response to external inputs, …)

Roles of Documentation In Engineering

There are two basic roles for documents, description and specification.

- ***Descriptions*** present properties of a product that exists (or once existed).
- ***Specifications***: descriptions that state only the required properties of a product.
- ***Full specifications:*** specifications that state all required properties.
  - Descriptions may include properties that are incidental and not requirements.
  - Products that don't satisfy a specification are not acceptable for the use intended.

The difference is one of intent, not form or content.

- Every specification that a product satisfies is also a description of that product.
- The notation can be the same.
- This has confused many computer science researchers.
- There is no such thing as a "specification language"!

If a product does not satisfy a specification, it is the product that is wrong.

If a description does not fit a product, it is the description that is wrong.

Specifications vs. Other Descriptions

Distinction is important when one product is used as a component of another.

- The builder of the using product may assume that any replacements will still have the properties stated in a specification.
- This is not true if the document is a description but not a specification.

Specifications impose obligations on the implementers, users, etc.

- When presented with a specification, implementers may either:
  - accept the task of implementing that specification, or
  - reject the job completely, or
  - report problems with the specification and propose a revision. (no "best effort")
- Users must be able to count on the properties stated in a specification;
- Users should not base their work on any properties not stated in the specification.
- Purchasers are obligated to accept, and pay for, any product that meets the (full) specification included in a purchase agreement or bid.

Models vs. Documents

Models are simplified versions of "the real thing".

- model airplanes
- linear model of a resistor
- atmospheric models for weather prediction
- axiomatic description of a stack

Some properties of a model may not be properties or the actual device.

Models are useful but must be used with care.

If you verify a model, you may still have flawed code.

Engineering documents are descriptions of the real thing.

What you can derive from a document should be true of the real thing.

The "use with care" warning is not needed.

Documents may be good models but will usually be difficult to analyse.

Why I Stress Documentation

Formal methods are often presented as "something extra".

Managers and developers know that they need documentation.

Developers and Maintainers know that today's documentation is (next to/worse than) useless because:

- not complete
- not consistent
- not always accurate
- hard to find the information that is there
- ambiguous
- hard to use for testing or use.

If we use mathematics properly, we can do better.

Not "in addition to" current documents but "instead of".

Role of Mathematics in Today's Software Development

Today

- Numerical Analysis is available but no longer taught in many CS programs
- Simple expressions appear in programs
- Simple Boolean reasoning used for conditional statements.
- Surveys have programmers denying that they use mathematics.
- They do not use the math that they learned in CS courses.
- They do not use math for documentation, design, and checking.

In the future…

- The subject of this talk
- We can use math in the ways that other Engineers use it, if we find the right Math.

Important Characteristics Of Engineering Mathematics

Mathematical structures are:

- Abstract: Definitions do not refer to the real-world objects that inspired them. Properties are formally stated without mention of real world behaviour.
- Static: Mathematical structures do not change with time.
- Precisely defined: Classical mathematical definitions are mature (well structured, well understood, widely accepted). All conclusions about the structures can be been derived from the definitions.

We shall refer to these three properties as the ASP properties.

- Abstraction allows reuse
- Static structures easier to study and apply.
  - Time handled as one more variable (dimension)
- Precise definitions avoid misunderstandings.

What Are Formal Methods

Formal methods **comprise**

- a notation (language) for describing, specifying or modelling computer systems
- procedures for using the language to study a computer system.

Mathematical Methods

- No special notation for computer systems
- Convert hypotheses about computer system to mathematical theorems.
- Theorems proven using standard mathematical techniques.

Comparison: Engineering Math vs. Formal Methods

Property	Engineering Math	Formal Methods
Nature of tool	Abstract, Application Independent	Models of programs that are programs
Dynamic Systems	Static: Time just another variable	Time has special treatment; models describe event sequences
Basis	Precise Definition	Incomplete definitions, many flaws
Maturity	Stable - minor variations	Many languages, frequent changes, many dialects
Application Method	Complete description of products	Partial description of products.
Reasoning	Evaluation of “closed form” solutions	Axiomatic derivation (requires search)

Abstractions and Their Representation

Mathematics is the study of abstract structures.

- ***Abstraction***: One description represents many differing things equally well.
- Structures are described by axioms, equations or other expressions.

A description of the an abstraction is called a *representation*.

One abstraction can have many representations. e.g.,

These are all representations of the same function

“f(x) = x+1”
f(w) = w+1
“f(x) = 1+x”
“f(z) = 2×z -z +1”

People confuse the abstraction with one of its representations.

“predicate” vs. “predicate expression”
“polynomial” vs. “polynomial expression”

What’s the best representation for software-implemented functions?

Mathematical Calculations

The main forms of mathematical "reasoning" are:

- Function evaluation
- Predicate satisfaction (e.g. equation solution)
- Inference or deduction

It is important to understand the difference and how each one is used in Engineering.

Function Evaluation

Given an expression that represents a function. e.g.,

"$f(z) = (2 \times z) - z + 1$"

Evaluation proceeds by

(1) substituting values for the variables

(2) evaluating functions

The order of evaluation is not fully determined but, when there is a choice, the result is not affected by the order chosen.[1]

No search is required.

[1] We are discussing mathematical expressions, not programs. There are no side-effects.

Predicate Satisfaction

Given an expression that represents a predicate. e.g.,

"$x = y+1 \wedge y>z \wedge \sin(x) < 1-\sin(z)$"

find values of the variables (x, y, z) that satisfy the predicate (evaluate to ***true***).

This requires function evaluation plus

- solving equations,
- simplification,
- search.

Solver (satisfier) is not assigned values; it assigns values.

Special case is solving "simultaneous equations".

Satisfying is generally more difficult than evaluation.

Engineers always prefer to find and use "closed form" solutions.

- "Closed form" solutions allow them to use only evaluation.

Inference or Deduction

Logic provides a sound basis for mathematics.

A logic (there are many variations) comprises

- a set of axioms (predicates assumed to be ***true***) and
- rules of inference (functions from predicate expressions to predicate expressions).

Anything inferred by applying the rules to the axioms and inferences is considered to be ***true***.

Proof procedures search for an inferences sequence that will demonstrate the truth of an hypothesis.

Inference requires

- evaluation
- search
- formula manipulation

Mathematical Reasoning in Engineering.

Engineers (and many others) use evaluation in their daily work.

In advanced applications, Engineers may solve equations.

In some modern applications they may use satisfaction and optimization search

Classical Engineers do not use deduction.

The Question:

Most “formal methods” are based on logic and inference.

- Is that part of the reason that they have not succeeded?
- Software cannot be described without predicate expressions, but
- Predicate expressions can be evaluated - we are not restricted to inference.

Applying Mathematics to Physical Systems

Variables in physics

- A measurable characteristic of a system whose value may change with time.
- Defined by describing how to measure, which gives meaning to the value of the variable.
- Variables usually identified by a short character string.
- Often, name or identifier not distinguished from the variable itself.

Variables in mathematics

- Variables are place holders used in the definition of relations.
- When used for this purpose, they do not have values.
- Variables have no meaning outside of the definition. For example

“doublesum(x,y) = x+y+ x+y”
is the same as
“doublesum (w, z) = w+z+ w+z”

Variable is usually represented by a short string, shorthand for “the nth argument”

Variable and its name often not distinguished

Applying Mathematics

Function Applications

- List of specific values for arguments (often called an assignment)
- doublesum(x,y) = x+y+x+y for (x, y) = (2, 3) = 2+3+2+3 = 10

Associating physical variables with mathematical variables by ‘punning’[1]

- Associate mathematical variables with physical variables that have the same name
- For example we may write

doublesum(length, width)

where ‘length’ and ‘width’ are the names of physical variables.

- The mathematical variables remain distinct from the physical variables
- Physical constraints on the values of the physical variables should be in the mathematics.
- In mathematical reasoning the association between the physical variable and the mathematical variable should not be used.

[1] I am indebted to H.D. Mills for pointing out the similarity of the same-sound pun with associating same-name variables.

How Engineering Mathematics Deals with Time

Time receives no special treatment in classical applied mathematics.

Time is treated in the same way as other physical variables.

Time is the fourth dimension of a 4 dimensional space

The first three coordinates form a conventional spacial coordinate system.

Pseudo Paradox:

Dynamic systems described by static mathematical structures.

Transfer Functions

A device's behaviour is characterized by a function with:

- a domain containing the possible histories of the observable behaviour
- a range containing the permitted value of the device's outputs.

Transfer functions describe the behaviour of the device without providing any information about its construction.

Transfer functions have a complete domain (all possible histories).

Given a network of devices, each described by its transfer function, it is possible to derive the transfer function of the network.

This supports a hierarchical documentation and analysis process.

- Analysis allows the detection of anomalous behaviour that might not be revealed by testing.

Non-functional relations may be used if some behaviour is not known or fully determined.

Applying Mathematics to Software Systems (I)

Variables and identifiers in programs

- Variables in programs are discrete state machines (FSMs) that serve as memory.
- The upper bound to the number of states may be very large and hard to characterize.
- Unlike variables in mathematics, program variables have a value (their state)
- The value can persist over time. Program variables are not placeholders.
- Program variables are closer to physics variables than the mathematical ones[1].
- In most programming languages:
  - It is possible for a variable to have several identifiers (aliasing).
  - One identifier may identify several different variables (block structure).
  - "Punning" can cause difficulties.
  - Aliasing (several names for one variable) is not easily avoided.

[1] Algol 60 introduced the distinction between variables (which were declared) and formal parameters (which were specified). Formal parameters were similar to the variables used in mathematics.

Assertion Based Methods (Best Known)

Robert Floyd (students Jim King and Zohar Manna)

- Predicates are associated with points in the program text
- Predicate transformation rules are given for each type of statement.
- Pre-condition transformed must imply post-condition assertion.
- Correctness of a program reduced to proving a set of mathematical theorems.

Looking at program proofs one can make the following observations:

- Some move often between program domain and mathematical domain. (not necessary)
- Abstraction is not supported.
- Some irrelevant variables are mentioned in expressions.
- If code is reused, proofs must be repeated.
- Precondition and postcondition treated as separate. Consequently, often necessary to add variables only to support the proof.
- The rules given are valid only if the relation between variables and identifiers is 1:1
- Program variable identifiers cannot appear as bound variables in predicate expressions.

Predicate Transformer Methods

- Dijkstra replaced the Hoare triple with a function from predicate expressions to predicate expressions, called a *predicate transformer.*
- He reversed the direction, i.e. start with postcondition to compute precondition.
- He took the possibility of non-determinism into account.
- Predicate transformers did not abstract from the state representation.
- They assume a 1:1 mapping between program variables and mathematical variables.

Abstract Relational (Functional) Methods

New concepts were not necessary; relation (from starting to stopping state) can be used.
It is necessary to introduce a way to denote non-termination.
This approach has a number of advantages:

- No necessity for new notation. This is not new mathematics; it is a new application.
- The definitions are independent of the way that the states and functions are represented.
- The definitions are simpler and, as a consequence, easier to understand.
- There is no need for a special treatment of time. Time is part of the state information
- The definitions are not tied to a specific set of statement types.
- The absence of specific primitive programs supports hierarchical application of the laws.
    - Constructed Programs can be treated as primitive.
- Relational methods are the closest software method to classical Engineering.
- No changes to “laws” are required to use new notations.

Additional views of software

Early work only considered individual sequential deterministic, terminating programs.
This was adequate in the 1960s but it is not adequate for the type of software we build today. F
Today we need to view software from many “orthogonal” viewpoints.

Each view can be described in a separate document as summarized below.

Document	Content
Software Requirements Document	Black Box specification, identifies all outputs and inputs and describes relation between output values and input history.
Module Guide	Informal description showing the hierarchical decomposition of the system into modules and the secret of each module.
Module Interface Specifications	Black Box specification, identifies all outputs and inputs and describes relation between output values and input history.” Usually shows externally invokable programs.
Module Implementation Design Document	Documents complete data structure, effects of all externally visible programs on data, abstraction function or relation (data interpretation) - design can be verified before coding begins.
Program Uses Structure	Description of permitted usage of one access-program by another. Determines the usable subsets of product.
Display Method Program Documentation	Hierarchical decomposition of program into “small programs” with specification of subject program and programs used.

Applying Mathematics For Other Views

No well-known formal method was designed with these views mind.
The communities behind these methods do not discuss a multi-view approach.
They try to use the same approach for all.
The relational approach, can be used for each of the views.
Most of the views can be documented by representing mathematical relations
The range and the domain of the relations varies with the view.
The exception is the module guide, which is the only informal document.
The next chart summarizes the relations represented in each of the documents.

Relational Content of Main Software Design Documents

Document	Relations	Domain	Range
Software Requirements Document	One per output variable: possible value after history	Histories of I/O	Output values
Module Interface Documents	One per output variable possible value after trace	I/O traces (sequences)	Output values
Module Implementation Design Documents		Declaration of complete data structure	
	Abstraction Relation: Trace lead to state	set of traces	Data Structure States
	Access Program Function (one per access program)	Data Structure States	Data Structure States
Program Uses Structure	Uses	Access Programs	Access Programs
Display Method Program Documentation	Program functions and program texts	program start states	program end states.

Mathematics gives us a way to say precisely what belongs in each document without specifying format or notation. Next, we must look at notation.

Representing Software Relations

Classical representations are not suitable for the functions that are described.

Functions describing physical products have a small number of discontinuities.

Functions that describe software have many discontinuities for 2 reasons:

- We are dealing with digital systems.
- Behaviour that has many special cases. (conditional statements)

Expressions describing computer systems very complex, hard to write and read.

The use of a tabular format for mathematical expressions can turn an unreadable string of symbols into an easy to access, complete and unambiguous document.

The next slide is an expression that describes the behaviour of a keyboard checking program that was developed by Dell in Limerick, Ireland

Example of Conventional Expression

Keyboard Checker: Conventional Expression

(N(T)=2∧keyOK∧(¬(T=_)∧N(p(T))=1))∨(N(T)=1∧(T=_∨(¬(T=_)∧N(p(T))=1))∧
(¬keyOK∧¬prevkeyOK∧¬prevkeyesc))∨((¬(T=_)∧N(p(T))=1)∧
((¬keyOK∧keyesc∧¬prevkeyesc)∨(¬keyOK∧keyesc∧prevkeyesc∧
prevexpkeyesc))∨((N(T)=N(p(T))+1)∧(¬(T=_)∧(1<N(p(T))<L))∧(keyOK))∨
((N(T)=N(p(T))-1))∧(¬keyOK∧¬keyesc∧(¬prevkeyOK∧prevkeyesc∧
preprevkeyOK)∨prevkeyOK)∧((¬(T=_)∧(1<N(p(T))<L))∨(¬(T=_)∧N(p(T))=L)))∨
((N(T)=N(p(T)))∧(¬(T=_)∧(1<N(p(T))≤L))∧((¬keyOK∧¬keyesc∧(¬prevkeyOK∧
prevkeyesc∧¬preprevkeyOK))∨(¬keyOK ∧¬prevkeyOK∧ ¬prevkeyesc)∨
(¬keyOK∧keyesc∧¬prevkeyesc)∨(¬keyOK∧keyesc∧prevkeyesc∧
prevexpkeyesc))∨((N(P(T)=Fail)∧(¬keyOK∧keyesc∧prevkeyesc∧
¬prevexpkeyesc)∧(1≤N(p(T))≤L))∨((N(P(T)=Pass)∧(¬(T= _)∧N(p(T))=L)∧(keyOK))

Characteristic Predicate of Keyboard Checker Program

The next slide is a tabular mathematical expression representing this function.

N(T) =

			T=_	¬ (T= _) ∧		
				N(p(T))=1	1<N(p(T))< L	N(p(T))= L
keyOK				2	N(p(T))+ 1	Pass
¬keyOK ∧	¬keyesc ∧	(¬prevkeyOK ∧ prevkeyesc ∧ preprevkeyOK) ∨ prevkeyOK			N(p(T)) - 1	N(p(T)) - 1
		¬prevkeyOK ∧ prevkeyesc ∧ ¬preprevkeyOK			N(p(T))	N(p(T))
		¬prevkeyOK ∧ ¬ prevkeyesc	1	1	N(p(T))	N(p(T))
	keyesc ∧	¬ prevkeyesc		1	N(p(T))	N(p(T))
		prevkeyesc ∧ ¬prevexpkeyesc		Fail	Fail	Fail
		prevkeyesc ∧ prevexpkeyesc		1	N(p(T))	N(p(T))

Tabular Expression for Keyboard Checker

Evaluating a Tabular Expression

The expression shown is a mathematical expression, evaluated in the usual way, substituting values and applying functions.

- It comprises 3 elements called *grids*.
- Each grid contains a number of *cells*.
- For this type of expression, the top grid and the left grid are called “headers”.
- The lower right grid is the “main grid”.

To use this type of tabular expression:

- Evaluates the cells the headers to find the ones that evaluate to ***true***.
- Exactly one of the column header cells and one of the row header cells will be ***true***.
- The indices of those cells identify a row and column.
- They identify one cell in the main grid.
- Evaluating the expression in that cell will yield the value of the function.

Why Tabular Expressions Make Mathematics Applicable

A tabular expression parses the conventional expression for the user.

Instead of trying to parse the complex expression, one looks at the simpler expressions that appear in the tabular expression.

Generally, one will only need to evaluate a few of the subexpressions (the expressions that appear in the cells) to compute an answer.

The user is evaluation functions, not using predicate satisfaction or inference.

Defining The Meaning of Tabular Expressions

Tabular expressions were successfully used without proper definition

Intuition is often enough, but...

Precise semantics are needed for tools and full analysis.

There have been five basic approaches to defining these expressions.

- Parnas constructed relations from component relations
- Janicki and his co-authors used information flow graphs
- Zucker based his definition on predicate logic.
- Khédri and his colleagues based their approach on relational algebra.

All three of these approaches were limited to basic forms of tables

- Jin and Parnas: translate tabular expression to conventional expression.
- General approach and provides a good basis for tools.

Types of Tabular Expressions

There are many forms of tabular expressions.

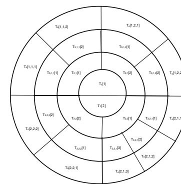
The grids need not be rectangular.

Not limited to two-dimensional grids or tables.

T_1

x + 1	y + 2	x + y	x - y	y - x
$V_1>1$	***true***	***true***	$\neg(V_1>1)$	$\neg(V_1>1)$
$\neg(V_2>5)$	$V_2>5$	$V_2>5$	$\neg(V_2>5)$	$\neg(V_2>5)$
true	$V_3+3>5$	$\neg(V_3+3>5)$	$V_3+3>5$	$\neg(V_3+3>5)$

T_0



Any expression in a cell can itself be a tabular expression.

It is possible to pick a form that fits the problem.

Good communication between user and implementer.

Promoting Discipline[1]

The software we use every day shows signs of lack of “Engineering Discipline”

Normal Cases work, but overlooked cases do not.

Mathematics provides a basis for discipline.

- Identify the output variables
- Identify the input variables
- Identify the domains
- One function per output
- Refer only to inputs
- Check for complete domain coverage
- Check for consistency.

[1]Parnas, Risks of Undisciplined Development, Comm. ACM, Vol. 53 Issue 10, October 2010, pp.25 - 27

Knowing When to Stop[1]

Most developers stop each phase too soon.

- They do not complete investigation and documentation of requirements
- They do not specify interfaces completely
- They do not make diagrams meaningful.
- They do not consider all error classes.
- They do not inspect enough.
- They overlook cases when testing.

Mathematics helps us to know when we have done enough

[1] Parnas,The Risks of Stopping Too Soon, Comm. ACM, Vol. 54, 6, June 2011,31-33

So, How Should Software Engineers Use Mathematics?

Apply math the way that other Engineers apply math

- Document the transfer functions
- Check for anomalies
- Test against what the mathematics predicts

But,...

- Use tabular expressions.
- Use discrete mathematics (predicate expressions)

That's all

Formal Methods, Who Needs Them?

Everybody and Nobody

Why Everybody

- Because we can all improve our performance and need to do so

Why Nobody

- Because we can use Engineering Mathematics, do not need FM

What Can We Do With Mathematics

Requirements Documentation

- Complete
- Consistent
- Precise
- Usable for testing
- Usable for simulation
- Checkable

Module Interface Documentation with the above properties

Program Function Documentation with the above properties

Module Design Documentation with the above properties

Highly Effective Inspections

Highly effective testing

Back To The Main Messages

1. **CS Formal Methods have not succeeded and will not succeed: Why?**
   - They are wrong. Not based on Engineering Mathematics.
2. **Engineering Mathematics is different from CS Formal Methods**
3. **Engineering Mathematics can be adapted for discrete functions**
   - Predicate Expressions, Tabular Expressions
4. **Mathematics can help those who build and maintain software**
   - Design
   - Document
   - Analyze
   - Test
   - Maintain
   - Certify